Virtual Proxy Servers for WWW and Intelligent Agents on the Internet

Sun Wu and Chang-Chain Liao

Department of Computer Science and Information Engineering
National Chung Cheng University
Chia-Yi, Taiwan, R.O.C.

Abstract

World-Wide Web (WWW) becomes extremely popular in recent years. However, there still exists many fundamental problems not yet solved in a satisfactory way, such as the information search, distribution, caching, filtering problems, etc.. Users still often suffers from long retrieval latency, and the lack of search function which can conduct localized search or hierarchical search. In this paper, we propose a framework for solving the above mentioned fundamental problems by introducing a new model of WWW proxy servers. In our model, *a proxy is not just an add-in to existed Web servers to provide cache and security functions, but works as a middle layer between browsers and Web servers, which can be plugged in a variety of functions for WWW applications, such as search engine, resource index, information filtering, robot, profile, intelligent agents, etc..* It's a framework for implementing hierarchical information servers and it's the right place to plug in various intelligent agents. To illustrate the usefulness of our model, we have implemented two fundamental functions into this middle layer : *the object lookup scheme (collaborative caching)* and *the searchable proxy.* We call this extensible proxy server *"Virtual Proxy"* .

Since search and retrieval is the most fundamental problems in World Wide Web, we will present this paper in a way to show how these two problems can be solved in a better way using the Virtual Proxy servers.

1 Introduction

The *World-Wide Web* (WWW) is a document distribution system based on a client/server model [3] . The client programs (also known as browsers) are used to receive information from servers. The Web took a hypertext/hypermedia approach, in which WWW documents could contain graphics, video, audio and many other different types of information. Documents are linked together through the hyperlinks. Clicking on a link signals the browser to fetch the desired document and then show it on the user's screen. Users follow the links to explore information easily. These attractive features make WWW extremely popular.

According to [5], for example, in the first ten months of 1994 the amount of WWW traffic on the Internet doubled roughly every 11 weeks. This increasing population incurs several problems. The first important problem is *the increasing document retrieval latency.* Obviously, when the number of clients accessing popular Web servers grows, the Web servers will be overloaded and the document retrieval latency will increase. Thus, the problem of how to reduce the latency is becoming very important.

WWW servers with caching proxy have been used to reduce the retrieval latency. If a user request for a page that has been cached in the proxy server, that request will be satisfied by the local copy instead of connecting to the original site to get that page. This mechanism will effectively cut down the network traffic if the cache hit ratio is high.

However, a WWW server with caching proxy alone might not solve the problem perfectly because: 1) The limited size of the cache will not guarantee a high hit ratio since the information amount is growing so fast. 2) Suppose we can enlarge the cache size arbitrarily to increase the hit ratio. Although the hit ratio can be increased, such approach may not work because there will be too many browsers to use such server and the bottleneck effect will jeopardize its efficiency.

One way of improvement is to make the caching proxies built up hierarchically, so that caching proxies can answer a request through the help from other caching proxies. Figure 1 shows an example of a hierarchical caching proxy tree.

The *Harvest* Cache [6] is a hierarchical Internet ob-

1060-3425/97 $10.00 © 1997 IEEE

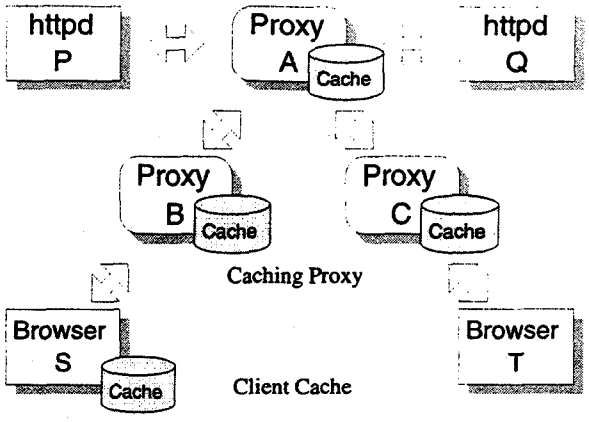

Figure 1: An example of a hierarchical caching proxy tree

ject cache system. When a caching server receives a request for a document that was not cached, it can call upon its parent and siblings in the hierarchical tree to inquiry about if any of them have cached the document. On the other hand, Malpani et al. [10] proposed a cooperative caching mechanism which uses IP multicast to communicate between the cooperative servers to locate the requested document. In our observations, *current object resolution mechanisms require real connections among the cooperative caching servers to locate the specific object. If the number of cooperative caching proxies is large, the increased connection overhead will make the approach inefficient.*

In this paper we will show that using our virtual proxy servers, we can achieve a collaborative caching effect where the virtual proxy servers in a virtual proxy tree would cooperate with each other to become a super scale cache. We implement a resource lookup mechanism in the virtual proxy tree through a resource index table called GRI (Global Resource Index) table which can be used to locate a requested object in the virtual proxy tree. Our approach has the advantage that the resource distribution information is maintained and that in finding a requested object, the virtual proxies only need to look up the tables instead of making real connections to siblings or cooperative servers as [6] or [10] does.

The other fundamental problem in the WWW is the *search problem.* Currently, there are search engines such as *Lycos, Alta-Vista,* to help users find wanted information. However, as such search engines do not provide localized search, sometimes the result to a query may all be links to sites far away from the users while the wanted pages actually exist in the proxy server of the inquiry user. Also, it's very often that a page consumed by one user in a community is also interesting to other users in the community. On the other hand, there exists situations where users may want to review or access documents which they saw before. So, the ability to do localized search is highly desired.

If we have search power that provide localized search or hierarchical search (when the local proxy server does not contain a match then search the parent proxy server), then the users would get more relevant result to their queries.

Warmlist [8], developed in University of Arizona, is a software to add the local search function to a user's local environment. However, its implementation depends on the browser's interface by monitoring `Mosaic's Hot List` file and replacing it with a named pipe. It is not suitable for other browsers like `Netscape` and the solution is too specific.

One question is how to implement the search functions in local environment and how to implement the search functions in a proxy server? A possible way is to add the search function by installing a search engine in the local machine and a monitor daemon to catch all the pages that are consumed by the users, such that those consumed pages can be searched later. On the other hand, the search function can be added to a proxy server through the WWW server's CGI interface since the proxy server is traditionally inside a WWW server.

However, these solutions are quite ad hoc, and we generally saw many solutions implemented in different architectures for different problems.

What we want, and what we think as the best solution is that we want all the information searching, retrieval, caching, filtering, distribution, or intelligent agent functions, etc., be implemented within a framework under one architecture. Our answer is to propose a new model for the proxy servers. *In our model, the proxy server is detached from the WWW server, and is enriched by adding the Common Gateway Interface capability to it. In our model, the World Wide Web contains three basic components, the WWW server, the Browser, and the Proxy server, where the WWW server is for information providing, the Browser is for information consumption, and all the above mentioned functions such as searching, retrieval, caching, filtering, distribution, etc., belongs to the Proxy servers.* In short, a proxy server is a middle layer between the Browser and the WWW server, which can be plugged in various functions such as searching, re-

trieval, caching, naming, filtering, intelligents agents, etc., for different kinds of WWW applications. We call this extensible proxy server *"Virtual Proxy Server"*. Figure 2 shows the virtual proxy as a middle layer between clients and servers.

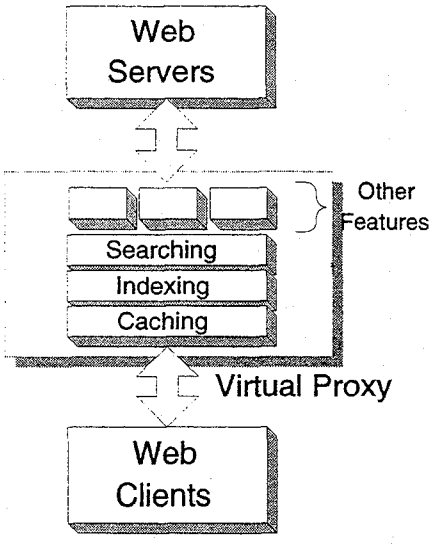

Figure 2: Virtual proxy as a middle layer between clients and servers

Virtual proxies could be put anywhere on the path connecting clients and servers. Users can set up a virtual proxy in their personal environments to provide caching, indexing, searching functions. A virtual proxy can be treated as a caching proxy to make its cache shared by multiple clients. Virtual proxies also can be structured in a hierarchical way such that for each virtual proxy in the hierarchical tree, it may have a parent and many siblings and it may cooperate with the other virtual proxies to provide collaborative caching and hierarchical searching. The one which has no parent is the root of the virtual proxy tree.

To show the usefulness of our model, we have implemented two basic functions in the virtual proxies. The object lookup scheme function is used to locate an object in the tree. A *global resource index* (GRI) table is created to maintain the object distribution information for the cooperative virtual proxies in the tree. *A virtual proxy can locate a requested object by looking up the GRI table without making real connections, as was done in* [10].

When a virtual proxy is put in the personal environment, it will cache all the documents requested by an individual user. By using the searchable proxy function, the user can search his cache to find out what he saw before easily. Meanwhile. When the virtual proxy is shared by a community, users can query the virtual proxy to find interesting documents requested by the other users in the same community, which can increase the usage of the information and reduce the network traffic.

This paper is organized as follows, in Section 2, we introduce the overall architecture of a virtual proxy. The object lookup scheme and the searchable proxy, together with their implementations on a virtual proxy, will be described respectively in Section 3 and Section 4. Conclusions and future work are remarked in Section 5.

A prototype of the *Virtual Proxy* server which was embedded with searching, data retriever (robot), and caching functions can be tested by setting the browser's proxy to 140.123.101.123 at port 9090.

2 The Overall Architecture of a Virtual Proxy

We use a hierarchical cache model similar to the one in the *Harvest* object cache system [6] to build up all the virtual proxies. Figure 3 shows an example of a virtual proxy tree. In the following, we will describe how the virtual proxies in the virtual proxy tree work together to improve the total performance.

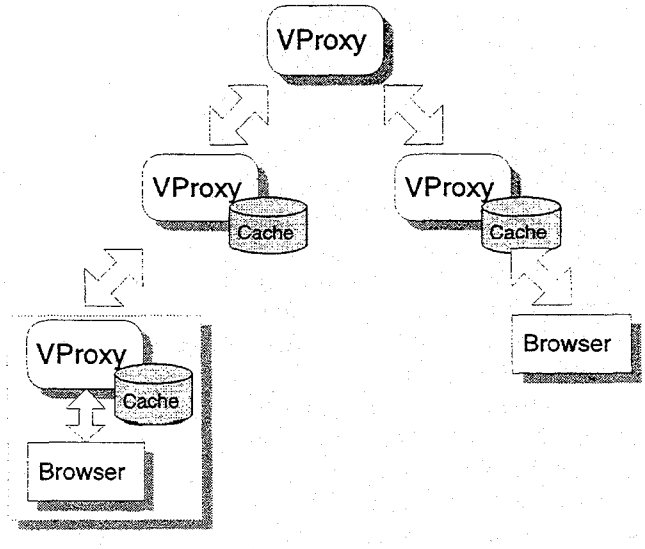

Figure 3: An example of a virtual proxy tree

The virtual proxy in the hierarchical tree is placed with the function *object lookup scheme*, which is used

to locate the exact position of the requested object in the hierarchical caching tree. When a virtual proxy receives a request for asking an object, it will look up its *global resource index* (GRI) table, which will be described in Section 3 in details, to check whether this object is in the current caching tree. Our schemed is different from the other approaches in that the object distribution information is kept through the Global Resource Index table, such that the finding of a cached object within a virtual proxy tree can be done by simply looking up the index table rather than *multicasting* the request to all the cooperative proxies or all its siblings.

In the hierarchical proxy tree, however, the root proxy could become overloaded if the tree is large. To avoid the bottleneck problem in the root proxy, the root proxy is in general designed to provide object lookup function only and does not physically cache objects passing through it.

To be able to plug in functions into the virtual proxy, we implement the Common Gateway Interface capability into the virtual proxy server. We define a new URI naming convention such that the CGI in the proxy server can be accessed. To provide the search function in the virtual proxy, we plug in the search engine *GAIS* [11], which is a general-purposed index/query system and design a set of CGI scripts to handle the query processing.

The built-in CGI capability in the virtual proxy also allows various functions to be plugged in for different WWW applications. As the virtual proxy server is the middle-way where the requested pages pass through, it's easy and is the right place to plug in information filters or intelligent agents. In the software we implemented, (the prototype can be tested by setting the browser's proxy to 140.123.101.123 at port 9090.) the proxy server would, for each page passes through it, add a function menu-bar in the beginning of the page such that the user can access the functions provided in the virtual proxy server by simply clicking on the respective function icon.

3 Object Lookup Scheme

The *object lookup scheme* is a dynamic lookup method for locating the requested object in the virtual proxy tree. The *global resource index* (GRI) table is created similarly to an inverted index for it maintains the relationship between an object and the set of virtual proxies where the object is stored. Each entry in the GRI table contains two parts: one is the URL of an object, and the other is a list of virtual proxies which have cached this object. For example, for an object X in the GRI table of a virtual proxy V, all the children of V which have cached the object will be listed in the entry corresponding to X. Besides, when an object is newly cached in the virtual proxy tree, the related proxies must update their GRI tables for consistency.

There are two types of virtual proxies used in the virtual proxy tree. One will cache objects, and the other won't. The later type has the object lookup function only and caches no objects. In the hierarchical caching model, a virtual proxy will send a request to its parent if it does not cache the requested object. When a virtual proxy P receives a request from its child V to specify an object X, it is assumed that V will cache the object X, and V's location is added to the entry corresponding to X in the P's GRI table. Yet, if the child V has no cache, the object will not be cached in V; therefore, it is not necessary to add V's location into the P's GRI table. Clearly, a virtual proxy can keep track of the information about which objects are stored in it and in its children, and where they are stored through its GRI table.

In the implementation, a new header `Vproxy-Location` is used by a virtual proxy. Note that the header should be only used in the request message. We define the syntax of `Vproxy-Location` as follows.

```
Vproxy-Location
= "Vproxy-Location" ":" vproxy-name
vproxy-name = dns-name ":" port-number
```

The `vproxy-name` field specifies the host `dns-name` and the port number of the virtual proxy which is sending this request. In the following is a correct header.

```
Vproxy-Location: gais.cs.ccu.edu.tw:9090
```

Figure 4 shows an example of building global resource index tables in a virtual proxy tree. When a virtual proxy receives a request message containing the `Vproxy-Location` header, it must add the `vproxy-name` into the proxy list of the corresponding entry in its GRI table. Surely, the virtual proxy does not need to update its GRI table when a request message does not contain the `Vproxy-Location` header.

Since an object may have many copies in the virtual proxy tree, looking up the object will return a list of proxies containing the object, but only one of them would be returned for the request. Our selection policy is always to choose the latest accessed proxy to return. The reason is that the copy in the latest accessed proxy would be the latest updated copy. In

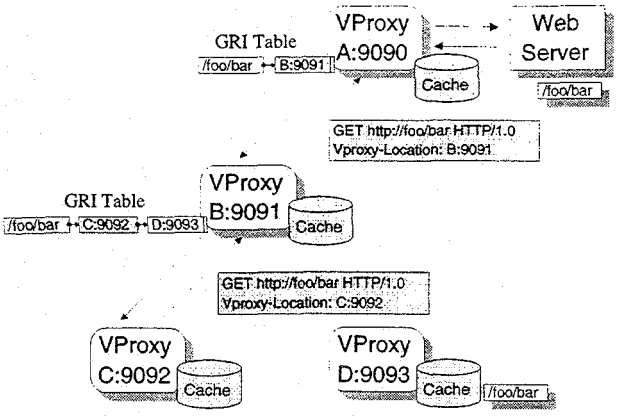

Figure 4: An example of building a global resource index table

addition, when a virtual proxy has cached an object, it will not send a request to its parent to get the object except when the copy has expired. This policy would result in a way of load sharing since the latest accessed proxy will serve the following request for the updated copy.

Now we describe the basic object lookup scheme in the virtual proxy tree. All the virtual proxies in the tree will maintain a GRI table to record the existence of the objects cached in it and in its children. In processing a request, a virtual proxy would check its GRI table to see whether such requested object is cached in this server or any of its children. If the requested object is cached in this virtual proxy, it then return the object. If the requested object is cached in some of its children, then it will process the request in a way as described in next paragraph. Otherwise, it will pass the request to its parent proxy if it is not the root virtual proxy. If current virtual proxy server is the root proxy and the requested object is not found in the GRI table, there are two cases: 1) if the root proxy has cache, it then fetches the requested object from the original site and return the object down the requesting chain; 2) therwise, it will return the URL through the indirection message to requesting child, in which case that child will take the responsibility of retriving the requested object from the original site. In any case, virtual proxies along the requesting chain would then update the GRI tables accordingly.

In case the current virtual proxy finds the requested object through its GRI table, it will get the location of the object from the corresponding entry in the table, retrieve the object, cache it, and return it to the requesting child; however, if a virtual proxy does not have a cache, (for example, the root virtual proxy that only provides object lookup function) and the requested object is in one of its children, it will return the child's location in a redirection message to the requesting child instead of getting the object by itself. When a virtual proxy receives a redirection response, it should be able to get the requested object from the redirection location.

The object lookup algorithm is shown as follows, which will execute when a virtual proxy receives a request asking for an object.

```
Let X denote the requested object;
Request_Handler {
    if ( X is in local cache ) {
        return the object from local cache ;
    } else if ( X is in the GRI table ) {
        get location of X from GRI table ;
    } else if ( parent vproxy exist ) {
        forward this request to parent ;
        wait for the parent response ;
        if ( response is a redirection ) {
            get object location from the
            redirection message ;
        } else if ( response is success ) {
            get the object from parent ;
            if ( caching is ON )
            cache the object ;
            return the object ;
        } else {
            return the message directly ;
        }
    } else {
        /* this vproxy is root; */
        set the object location to the
        original server ;
    }
    if ( caching is ON ) {
        get the object from the location ;
        cache the object in the cache ;
        return the object ;
    } else {
        return the redirection message ;
    }
}
```

Figure 5 shows an example of this lookup algorithm. The virtual proxy B, which provides the caching function, gets the object `/foo/bar` from its child F, caches it and then returns it to the requesting proxy C . Figure 6 shows another example of the lookup algorithm. The virtual proxy A, which does not have a cache, only returns a redirection message with the information of child G's location to the requesting proxy B. The proxy B gets the object `/foo/bar` from its sibling

G and returns it to its child C .

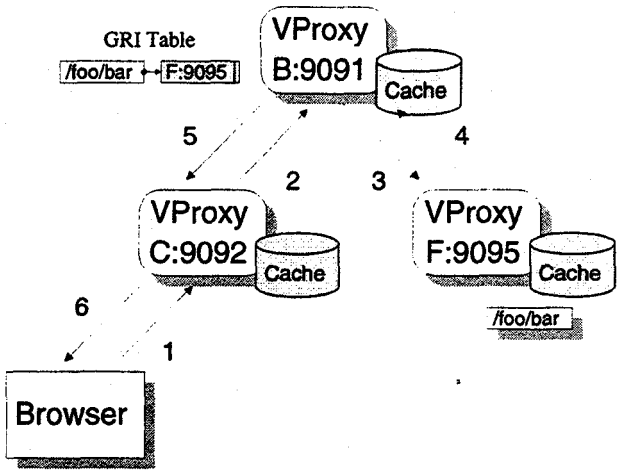

Figure 5: An example of the virtual proxy lookup algorithm

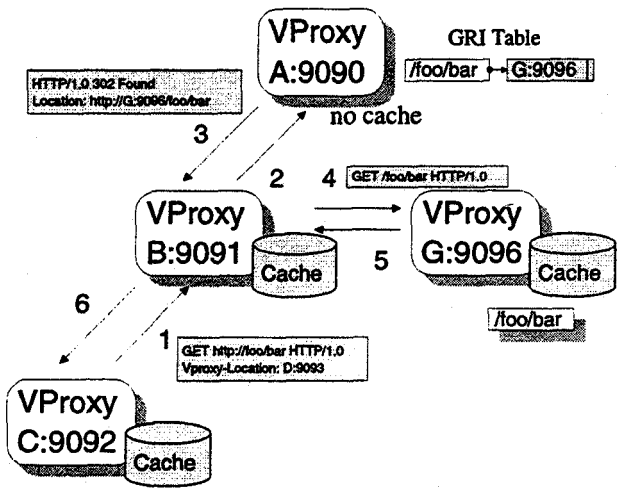

Figure 6: Another example of the virtual proxy lookup algorithm

Implementing the lookup algorithm, we use the response header field `Location` in our redirection message to indicate the location of the desired object in the virtual proxy tree. In Figure 6, the redirection message is :

```
HTTP/1.0 302 Found
Location: http://G:9096/foo/bar
```

The location `http://G:9096/foo/bar` is an absolute URI, which indicates the object `/foo/bar` is cached in the virtual proxy G with port 9096. When the virtual proxy B receives this message, it connects to the virtual proxy G through port 9096 to get the object `/foo/bar`. Note that proxies only accept the request in an absolute URI format. Thus, when a virtual proxy receives a request asking for an object in its absolute path which is a relative URI format, it should directly retrieve the object from its cache. The path name `/foo/bar` means that the object is cached from host `foo` and the file name is `bar`.

One case we have to be careful is when the virtual proxy receives a request for an object which was just removed from the cache by the cache manager. In such case, this virtual proxy will get the object directly from the original server. We must notice that the virtual proxy can not forward the request to its parent for avoiding an infinite loop of requesting. After the virtual proxy gets the object from the original site, we do not need to update the GRI table of its parent because this virtual proxy just caches the object again.

A variation of the object look up algorithm is to extend the GRI table of a virtual proxy such that it records the index, for each object, all the descendants of the virtual proxy which have cached the object. By this way, a virtual proxy will receive a response message with the header `Vproxy-Location` where a set of descendant virtual proxies caching the requested object are listed. The advantage of this approach is that now the effective cache size is the union of all the proxy caches in the proxy tree so there is more chance that a request will be satisfied through the cached object somewhere in the proxy tree. However, this approach is more complicated and the overhead of maintaining the GRI table will be increased. Moreover, maybe we need to define more new headers for passing some considerations such as the traffic cost, load distribution, or user's preference to its parent for choosing the most suitable proxy from the candidate list.

4 Proxy CGI and Searchable Proxy

A proxy server with cache function would store pages that were requested in the near past. If the proxy server is in the local machine, it means that all the pages viewed by the user are stored there. On the other hand, if the proxy server is in a community machine shared by many users, it would have stored those

pages that have been viewed by users in such community. It's very often that a page consumed by one user in the community would also be interesting to other users in the community, and it's often that a user would like to review a page he/she has seen before. If we could add search function to the proxy servers, it would allow the users to get more relevant results to their query since such search function can perform localized search or hierarchical search (when the local proxy does not contain a match, search the parent proxy server) which is not available in global search engines.

But the problem is how would we implement the search function in the proxy server. Our solution is to implement the *Common Gateway Interface* function into the virtual proxy server such that it can be easily plugged in functions into the virtual proxy through the CGI interface.

However, traditionally, CGI is provided in Web servers only, and proxy server is only regarded as a place for doing cache or for routing the requests. Therefore, the first problem met on implementing CGI in the virtual proxy is : *how can we indicate that we want to access the CGI in a virtual proxy ?*

The solution is to make a new URL naming convention.

For a browser or a virtual proxy, we use "." in the host field of URL to indicate its parent virtual proxy. For example, assumed that a proxy wants the CGI program `webgais` under the directory `cgi-bin` in its parent virtual proxy, it should use the URL as `http://./cgi-bin/webgais`.

Note that when the browsers do not set up their proxies to be virtual proxies, this URL convention may cause an error message. For example, if we open the URL `http://./cgi-bin/webgais` in a browser which does not set up its proxy to be a virtual proxy, the error message looks like :

```
ERROR: The requested URL could not be

retrieved While trying to retrieve the

URL: http://./cgi-bin/webgais

The following DNS error was encountered:

     ERROR 102 -- DNS name lookup failure

This means that:

     DNS Domain '.' is invalid:

     Host not found (authoritative).
```

Once the virtual proxy server is equipped with the CGI capability, it's easy to add in the search function to it. What we do is to add the *GAIS* [11] search engine through the CGI interface. To query the virtual proxy, a user should open a URL `http://./` to fetch the query fill-out form first. After the query form is filled out and submitted, a CGI script in the virtual proxy called the query handler will handle the query to generate a query result. Users can check the query result and examine which documents are needed, and then they can click the hyperlinks to get the documents from the virtual proxy cache.

Figure 7 shows an example of a query fill-out form. Basically, there is a text field in which user can input their query pattern. There are some query options which can help users specify what they want more clearly. We provide the following query options : *case sensitive/insensitive matching, word-base/substring matching, error-tolerant matching and Chinese-homophonetic matching* . Users also can specify the matching domain in the HTML documents. The matching domain could be the title field ($< TITLE > \ldots < /TITLE >$), the header field ($< Hx > \ldots < /Hx >$, where $x = 1..6$), the hyperlink text field ($< A\ HREF = \ldots < /A >$), or anywhere except the HTML tag fields ($> \ldots <$). Click on the `search` button for sending the query to the virtual proxy.

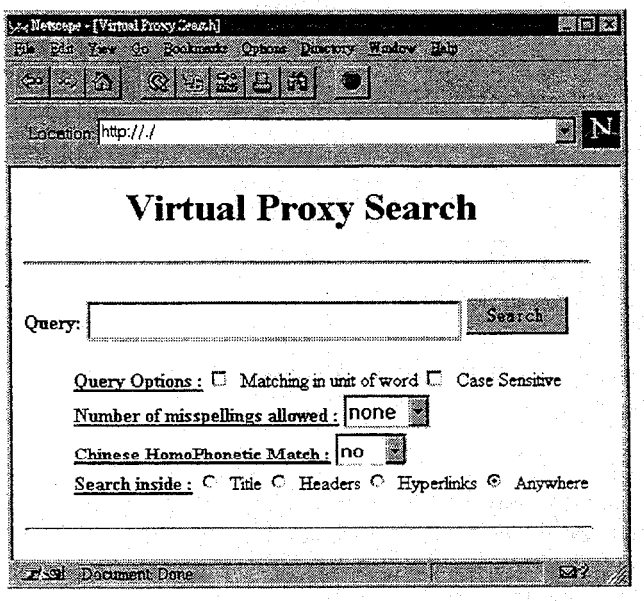

Figure 7: An example of a virtual proxy query form

Receiving the query, the virtual proxy invokes the

query handler to process the query in the following steps. Figure 8 shows the framework of the query processing.

Step 1 get the query information from the CGI interface, such as environment variables or data from standard input `stdin`.

Step 2 check user's input carefully and report error message if the input contains unknown format.

Step 3 translate the query pattern and the query options into the `gais` command line options, then generate an argument list for calling `gais` . Note that `gais` is the query processing program, a part of `GAIS`.

Step 4 execute `gais` without going into the shell.

Step 5 receive the `gais` output from pipe, generate the query result in HTML format, then send the query result to the browser.

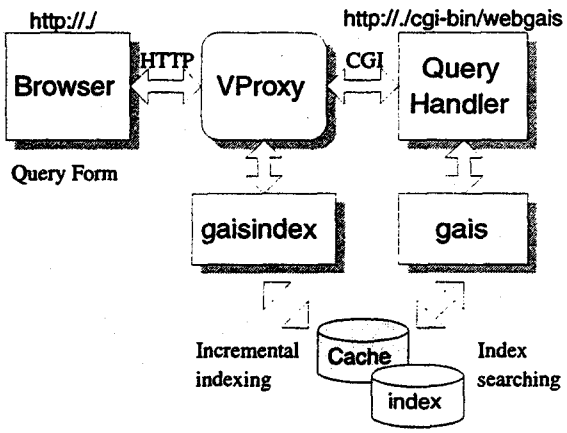

Figure 8: The framework of the virtual proxy query processing

Figure 9 shows an example of the query result. We also provide a pseudo stateful feature to help users view the query result in more efficient way. Every time the query handler returns a query result which contains only ten matched documents. When users want to see more documents, they can click `more` button to fetch next ten matched documents. This feature can save the network traffic load compared to all or a bunch of matched documents dumped to the user when the query matched a lot of pages. To implement the pseudo stateful feature, a temporary file is used to store the matched file list for each query. To fetch next ten files, users click on `more` button, which will send an identification of the matched file list together with the offset of the next ten files in the list, to the virtual proxy. The query handler gets the next ten matched files from the list and sends back the result to the user. We can delete these temporary files periodically by a cron job.

Besides, we output the matched documents with a simple ranking scheme where we define documents matching the query in the title field have a higher ranking than those matching in the header fields. Documents which match the query in the header fields have a higher ranking than those matching in anywhere.

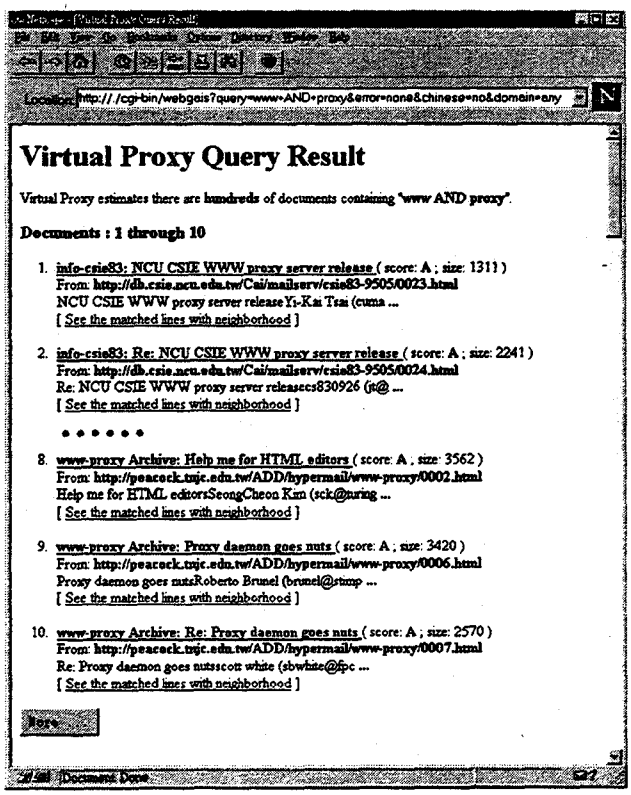

Figure 9: An example of a virtual proxy query result

To examine the matched lines within the matched documents also can help users to decide whether the matched document is really they want. Figure 10 shows an example of the matched lines of a document for a query.

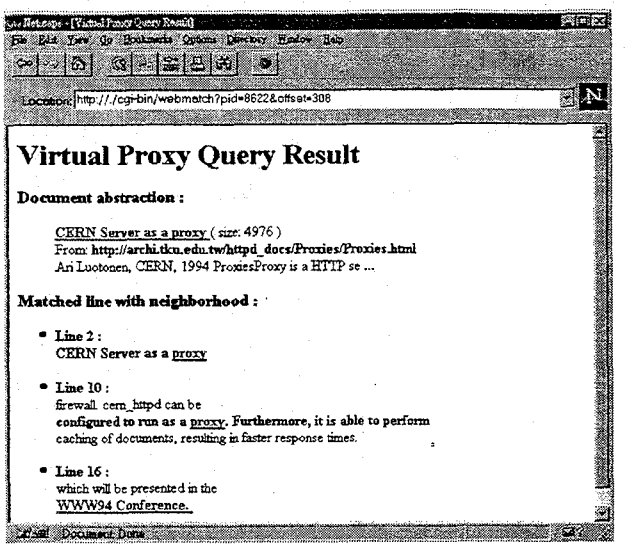

Figure 10: An example of matched lines for a query

5 Conclusions and Future Work

World-Wide Web (WWW) becomes extremely popular in recent years. However, there still exists many fundamental problems not yet solved in a satisfactory way, such as the information search, distribution, caching, filtering problems, etc.. Users still often suffers from long retrieval latency, and the lack of search function which can conduct localized search or hierarchical search. In this paper, we propose a framework for solving the above mentioned fundamental problems by introducing a new model of WWW proxy servers. In our model, *a proxy is not just an add-in to existed Web servers to provide cache and security functions, but works as a middle layer between browsers and Web servers, which can be plugged in a variety of functions for WWW applications, such as search engine, resource index, information filtering, data retriever (robot), profile, intelligent agents, etc..* It's a framework for implementing hierarchical information servers and it's the right place to plug in various intelligent agents. To illustrate the usefulness of our model, we have implemented two fundamental functions into this middle layer : *the object lookup scheme (grouped caching)* and *the searchable proxy.* We call this extensible proxy server *"Virtual Proxy"*.

In this paper, we have shown the usefulness of the Virtual Proxy server by implementing two basic functions into the virtual proxy : *object lookup scheme* and *searchable proxy.*

The key point of the object lookup scheme is to locate objects efficiently without making unnecessary connections via the maintenance of the *Global Resource Index* tables, which record the object distribution information in a virtual proxy tree. The object lookup scheme can effectively make the virtual proxies in the proxy tree cooperate with each other to become a super scale cache to increase the cache hit ratio and reduce the access latency. Moreover, the object distribution information maintained by the GRI table can be used to do the information distribution control as well as the cache consistency control in a more feasible way.

The other important function is the searchable proxy, which searches the cached documents in the virtual proxy cache. Such a virtual proxy can provide a personal Internet information environment for individuals, such that a user can search its personal cache to recall the documents which they have seen before easily. Users also can query a community virtual proxy to find interested documents which have been requested by the other people in the same community, since people in the same community may have some common interests.

For the future work of this research, there are four major directions to proceed: 1) conduct experiments to examine the performance of the collaborative caching which use our object lookup schemes based on the GRI index; 2) design new protocols between virtual proxy servers such that the information distribution control, replication control, object synchronization, etc., in the World Wide Web can be solved in a better way; 3) add more functions to the virtual proxy servers. 4) develop new software systems such as PIE (Personal Information Engine) based on the virtual proxy framework.

We plan to release softwares based on the virtual proxy framework in the near future, possibly through the following URL, http://gais.cs.ccu.edu.tw.

We thank Dr. Herman, Rao for many fruitful discussions. We thank Hui-Min Tsai for her kindly help on the writing of this paper.

References

[1] Alta Vista home pages. Digital Equipment Corporation, 1996. Available from http://altavista.digital.com/.

[2] Lycos home pages. Lycos Inc., 1996. Available from http://www.lycos.com/.

[3] Tim Berners-Lee, Robert Cailliau, Jean-Franis Groff, and Bernd Pollermann. World-Wide Web: The information universe. *Electronic Networking: Research, Applications and Policy*, 2:52–58, 1992. Available from ftp://ftp.w3.org /pub /www /doc /Article_9202.ps.Z.

[4] Azer Bestavros, Robert L. Carter, Mark E. Crovella, Carlos R. Cunha, Abdelsalam Heddaya, and Sulaiman A. Mirdad. Application-Level Document Caching in the Internet. In *Workshop on Services in Distributed and Networked Environments*, 1995. Available from ftp://cs-ftp.bu.edu /techreports /95-002-web-client-caching-ps.Z.

[5] Hans-Werner Braun and Kimberly Claffy. Web traffic characterization : an assessment of the impact of caching documents from NCSA's web server. In *Second International World-Wide Web Conference*, October 1994. Available from http://www.ncsa.uiuc.edu /SDG /IT94 /Proceedings /Dday /claffy /main.html.

[6] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A Hierarchical Internet Object Cache. Technical Report CU-CS-766-95, Department of Computer Science, University of Colorado, April 1995. Available from ftp://ftp.cs.colorado.edu /pub /cs /techreports /schwartz /HarvestCache.ps.Z.

[7] Peter B. Danzing, Michael F. Schwartz, and Richard S. Hall. A case for caching file objects inside internet-works. In *ACM SIGCOMM 93 Conference*, pages 239–248, 1993.

[8] Paul Klark and Udi Manber. Developing a Personal Internet Assistant. Technical report, Department of Computer Science, University of Arizona, December 1994. Available from http://glimpse.cs.arizona.edu:1994 / paul /warmlist /paper.html.

[9] Ari Luotonen and Kevin Altis. World-Wide Web Proxies. In *First International World-Wide Web Conference*, May 1994. Available from http://www.lsu.edu /internet /guides /www-docs /WWW /Proxies /Overview.html.

[10] Radhika Malpani, Jacob Lorch, and David Berger. Making World Wide Web Caching Servers Cooperate. In *Fourth International World-Wide Web Conference*, December 1995. Available from http://www.w3.org /pub /WWW /Journal /1 /lorch.059 /paper /059.html.

[11] Sun Wu, Pai-Hsin Hsu, Fu-Kai Ye, Bwo-Shong Hwang and Meng-Dar Liu, gais/gaisindex 1.0 – A General-Purposed Index/Query Engine. Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan, R.O.C., 1995. Available from http://gais.cs.ccu.edu.tw/.

[12] Lincoln D. Stein. The World Wide Web Security FAQ. Whitehead Institute/MIT Center for Genome Research, September 1995. Available from http://www-genome.wi.mit.edu /WWW /faqs /www-security-faq.html.